**4CS001 - Coding Challenge 3***Functions and File Handling*

**Due: Sunday 17th November 2019 at 11:59am**  
**This assignment is worth 20% of the overall module grade**

Task Overview: 2

Getting Started: 2

Requirements: 3

Step 1 – Loading the Words File: 3

Implementation Details: 3

Example Usage: 4

Step 2 – Identifying Unused Letters: 4

Implementation Details: 4

Example Usage: 4

Step 3 – Testing the Win Condition: 5

Implementation Details: 5

Example Usage: 5

Step 4 – Tracking Players Progress: 5

Implementation Details: 5

Example Usage: 6

Step 5 - The Main Game Logic: 6

Implementation Details: 6

Scoring: 7

Example Implementation (user input in red): 7

Testing Your Program 9

Examples: 9

Run Without Errors: 9

Run with Errors: 9

Challenge (Additional Credit): 10

Implementation Details: 10

Hints: 10

Example Implementation (input in red): 11

Introduction

This coding challenge will assess your knowledge of Python functions, file handling operations and exceptions. It also builds on many of the topics we have covered over the course of the module.

The marking scheme for this task is on Canvas, make sure you check back on a regular basis as you work through the assessment. **There is an additional challenge segment that can earn you extra credit, which you will need to complete to achieve the highest possible mark.   
Students who do not attempt this cannot achieve grades above 80% for this task.**

Task Overview

Your task is to implement a variation of the classic word game Hangman, which involves players guessing the letters in a word chosen at random with a finite number of guesses. While there are alternate versions such as category Hangman and Wheel of Fortune, which involve players guessing idioms, places, names and so on, we will be sticking with the traditional version.

If you are unfamiliar with the rules of the game, please read the following before starting: http://en.wikipedia.org/wiki/Hangman\_(game). You can also play an online version [here](http://www.hangman.no/). Don’t be intimidated ­ You’ll be given some skeleton code to get you started, it's easier than it looks!

Getting Started:

Start by downloading the files hangman.py, hangman\_tests.py and words.txt from Canvas, **saving them both in the same folder**. Start by looking through the main Python file and reading the included documentation. Add your name and student number to the top of the file.

Requirements:

You will implement a function called main that allows users to play an interactive hangman game against the computer. The computer should pick a word, and players should then try to guess letters in the word until they win or run out of guesses.

Here is the overarching behaviour we expect:

1. The program should load a list of available words from the text file provided. **Note that the file you have been given contains words in lowercase.**
2. The computer should then select a word at random from the list.
3. The user is given a certain number of guesses at the beginning.
4. The game is interactive; the user inputs their guess and the computer either:
   1. reveals the letter if it exists in the secret word
   2. penalizes the user and updates the number of guesses remaining
5. The game ends when the user guesses the word, or runs out of available guesses.

You are going to make use of a common approach to computational problem solving, which involves breaking the problem down into several logical subtasks to make things more manageable. The file hangman.py contains several existing helper variables and functions which you should use while developing your implementation.

Step 1 – Loading the Words File:

Implementation Details:

You should start by implementing the load\_words function, which should return a list of valid words (strings) from the text file you have been given. Implementation guidelines:

1. Print a message to let the user know that the word list is being loaded, this can sometimes take some time depending upon the size of the file and the speed of the CPU.
2. Open words.txt and read the contents into a variable. The words in the file are delimited (separated) by a single space.
3. Ensure that any exceptions raised when attempting to open the file are handled such as the file not existing or having the wrong permissions. If an exception is encountered, print a message to the user identifying the problem and end the program.
4. Print the total number of words loaded. There should be 55900 in words.txt.
5. Return the wordlist from the function.

You should test that your implementation works before moving on. Make sure that the correct number of words are loaded and that no exceptions are raised.

**Hints:**

* You can use the split function to separate a string into a list of strings.
* Make sure you close the text file once you’ve loaded the word list.

Example Usage:

>>> load\_words()  
Loading word list from file: words.txt  
55900 words loaded.

Step 2 – Identifying Unused Letters:

Implementation Details:

Your next task is to implement the get\_remaining\_letters function. This will be used to generate a string comprised of letters that have not yet been guessed. The function takes a single argument letters\_guessed, a list of letters (strings) that the user has previously guessed. You will need to write additional code to compare these letters against the full alphabet to determine the letters that remain and return them as a string.

**Hint:** You may find the string.ascii\_lowercase variable useful, which generates a list of alphabetical letters in lowercase. The string library has been imported for you.

>>> from string import ascii\_lowercase  
>>> print(ascii\_lowercase)  
abcdefghijklmnopqrstuvwxyz # All lowercase letters

Example Usage:

>>> letters\_guessed = ['a', 'e', 'c']  
>>> print(get\_remaining\_letters(letters\_guessed))  
bdfghijklmnopqrstuvwxyz # Letters minus letters\_guessed

Step 3 – Testing the Win Condition:

Implementation Details:

Your next task is to implement the is\_word\_guessed function. This should check whether the user has guessed all the letters in the secret word chosen by the computer and can be used to determine whether the game has been won. The function takes 2 arguments:

1. word - the word that the player is attempting to guess.
2. letters\_guessed - the letters that the player has previously guessed.

You will need to write additional code to determine whether the letters in word are present in letters\_guessed. Remember that when the player guesses a letter, all instances of that letter in the secret word are revealed to the player. The function should return a Boolean value (True/False) based on whether the letters in word are present in letters\_guessed.

**Hint:** Iteration and use of sets are both viable options for this step.

Example Usage:

>>> letters\_guessed = ['q', 'v', 'd', 'e', 'n', 'u']  
>>> print(is\_word\_guessed('queen', letters\_guessed))  
True # All of the letters are in letters\_guessed

>>> letters\_guessed = ['q', 'v', 'd']  
>>> print(is\_word\_guessed('queen', letters\_guessed))  
False # Not all of the letters are in letters\_guessed

Step 4 – Tracking Players Progress:

Implementation Details:

Your next task is to implement the get\_guessed\_word function. This should return a string, comprised of letters, underscores (\_), and spaces that represents which letters have been guessed so far. The function takes two arguments:

1. word - the word that the player is attempting to guess.
2. letters\_guessed - the letters that the player has previously guessed.

You will need to write additional code to generate and return a string containing the letters that have previously been guessed and placeholders for those that haven’t. In this case you should use an underscore followed by a space (\_ ). Other symbols could also be used, but this is easily recognisable. You may be wondering why the space is required. The reason is that if several consecutive letters are hidden, it would not be possible to discern between them, imagine seeing this: M\_\_\_\_\_, it would be difficult to determine the number of remaining letters. However, with a space added, it is much easier to tell what is going on: M\_ tt\_ e\_ .

Example Usage:

>>> letters\_guessed = ['q', 'v', 'd']  
>>> print(get\_guessed\_word('queen', letters\_guessed))  
q\_ \_ \_ \_ # Only q is revealed as the other letters are incorrect

>>> letters\_guessed = ['q', 'e', 'd']  
>>> print(get\_guessed\_word('queen', letters\_guessed))  
q\_ ee\_ # When e is guessed, both occurrences are revealed

Step 5 - The Main Game Logic:

Implementation Details:

Now it’s time to tackle the main function, the part of the program responsible for starting and running your interactive hangman game. The function takes a single argument: word, which is used to store the secret word that the player is trying to guess. Make sure to use the functions you implemented in previous steps to tie the application together.

Begin by printing a welcome message to the player followed by the number of letters in the secret word, you should consider defining a welcome\_message function to do this.

You should use a loop to drive the main game logic. The game should consist of several ‘rounds’ in which the player guesses a new letter in the secret word. The game should end when the player guesses the word or uses all their available guesses.

In each round, you should:

1. Check if the player has guessed all the letters in the secret word. If so, print a message congratulating the user, and another to reveal their score (see below). End the game.
2. Check the players number of remaining guesses. You should either:
   1. Print a message informing the player they have run out of guesses. Print the secret word and end the game.
   2. Proceed to the next step if the player has 1 or more guesses remaining.
3. Print the number of guesses the player has remaining, traditionally players start with 6 guesses. However, you may wish to change this.
4. Print the remaining letters that the player has not yet guessed.
5. Prompt the user to enter a new guess. Uppercase and lowercase letters should be accepted.
6. Check if the guess is a valid alphabetic letter. You should either:
   1. Print a message informing the player their guess was invalid. Decrement the players remaining guesses by 1 and skip to the next round.
   2. Proceed to the next step if the guess is valid.
7. Check if the guess is the same as a previous one. You should either:
   1. Print a message informing the user they have repeated a previous guess and skip to the next round. Do not penalise the player for repeated guesses.
   2. Proceed to the next step if the guess is not the same as a previous one.
8. Give the user feedback on their guess. You should either:
   1. Print a message congratulating the player on a successful guess and display the partially guessed word with the appropriate letters revealed.
   2. Print a message informing the player of an incorrect guess and display their current progress. If the guess was a vowel decrement the players remaining guesses by 2 otherwise decrement the players remaining guesses by 1.
9. Print a separator such as ‘------------’ or an alternative to indicate the end of a round.

Scoring:

Users should receive a score if they win the game. The score should be calculated by multiplying the players remaining guesses by the number of unique letters in the secret word.

Example Implementation (user input in red):

Loading word list from file...  
55900 words loaded.  
Welcome to Hangman Ultimate Edition  
I am thinking of a word that is 3 letters long  
-------------  
You have 6 guesses left.  
Available letters: abcdefghijklmnopqrstuvwxyz  
Please guess a letter: a  
Good guess: a\_ \_  
------------  
You have 6 guesses left.  
Available letters: bcdefghijklmnopqrstuvwxyz  
Please guess a letter: e  
Good guess: a\_ e  
------------  
You have 6 guesses left.  
Available letters: bcdfghijklmnopqrstuvwxyz  
Please guess a letter: e  
Oops! You've already guessed that letter: a\_ e  
------------  
You have 6 guesses left.  
Available letters: bcdfghijklmnopqrstuvwxyz  
Please guess a letter: r  
Oops! That letter is not in my word: a\_ e  
------------  
You have 5 guesses left.  
Available letters: bcdfghijklmnopqstuvwxyz  
Please guess a letter: o  
Oops! That letter is not in my word: a\_ e  
------------  
You have 3 guesses left.  
Available letters: bcdfghijklmnpqstuvwxyz  
Please guess a letter: g  
Good guess: age  
------------  
Congratulations, you won!  
Your total score for this game is: 9  
  
**Example Loss:**

------------  
Sorry, you ran out of guesses. The word was: age

Testing Your Program

You do not need to look at or understand the hangman\_tests file. However, it can be used to test the functions in your hangman program. To do this simply run the file (make sure it’s in the same folder as your other files) e.g. python hangman\_tests.py

This will give you feedback letting you know if there are problems with any of your functions (excluding the main function – testing input/output is difficult!). You should run this program each time you finish writing one of the functions to see if it’s working as intended.

Do not change the name of your program file or edit the code in the tests file as this will prevent the program from running as expected and will generate errors.

Examples:

Run Without Errors:

>>> python hangman\_tests.py

---------------------------------------------------------------

Ran 4 tests in 0.013s

OK # All our tests passed without errors!

Run with Errors:

>>> python hangman\_tests.py

===============================================================

FAIL: test\_is\_word\_guessed (\_\_main\_\_.TestHangman) (test=Test(input=['q'], output=True))

---------------------------------------------------------------

AssertionError: False != True

---------------------------------------------------------------

Ran 4 tests in 0.008s

FAILED (failures=1) # 3 of our tests passed, but one failed!

This output might look a little confusing. However, it is telling us that there is a problem with the test\_is\_word\_guessed function. Its input was a list containing the letter 'q' for which the output was expected to be True. However, the value returned by the function was False. You can use information like this to track down problems in your logic and fix them.

Challenge (Additional Credit):

Currently, there isn’t much incentive for players to keep coming back to the game. There is no real progression system as players scores aren’t stored on a persistent basis.

Implementation Details:

For this challenge, you will extend the existing program so that players scores are saved to and read from a text file (scores.txt). You should ask for the player’s name at the start of each new game and use it to load their previous high score. If they manage to beat their personal best while playing, ask them if they would like to update the leaderboard and do so if prompted.

To do this, you should create two new functions:

* A function to retrieve the existing high score for the current player called: get\_score. This should take a single argument, the players name and return their high score. If they don’t have a high score return 0 or None. Players without a previous high score should always be asked whether they would like to store their score.
* A function to save a new high score for the current player called: save\_score. This should take two arguments (the players name and score). The function does not need to have a return value, but should handle writing the latest score to the text file and closing it. If a player already has a lower score saved in the file, update it instead.

Your game will also need a more robust menu, it should contain options to: play the game, load the leaderboard or quit. Players should return there after games or viewing the leaderboard.

Hints:

* When the player loads the leaderboard, you will need to read the text file used to store the data and present the information in a tabular format so that it’s easy to read.
* Selecting the option to quit should close the program after printing a goodbye message.

Example Implementation (input in red):

Loading word list from file...  
55900 words loaded.  
Welcome to Hangman Ultimate Edition

Do you want to Play (p) view the leaderboard (l) or quit (q): p

Score Name  
-----------------------------

7 Matthew Howell

5 Herbert Daly

3 Hiran Patel

Would you like to play (p) or view the leaderboard (l): p

What is your name: Matthew Howell

I am thinking of a word that is 2 letters long  
-------------  
You have 6 guesses left.  
Available letters: abcdefghijklmnopqrstuvwxyz  
Please guess a letter: a  
Good guess: a\_   
------------  
You have 6 guesses left.  
Available letters: bcdefghijklmnopqrstuvwxyz  
Please guess a letter: n  
Good guess: an  
------------  
Congratulations, you won!  
Your total score for this game is: 9

A new personal best! Would you like to save your score(y/n): y

Ok, your score has been saved.

Do you want to Play (p) view the leaderboard (l) or quit (q): q

Thanks for playing, goodbye!